ITL.1028US
(P16764)

# APPLICATION

# FOR

# UNITED STATES LETTERS PATENT

**TITLE:** SWITCHING PROCESSOR THREADS DURING LONG LATENCIES

**INVENTOR:** MICHAEL W. MORROW

Express Mail No. EV 337934335 US

Date: September 12, 2003

Prepared by: Trop, Pruner & Hu, P.C., Mark J. Rozman
8554 Katy Freeway, Ste. 100, Houston, TX 77024
713/468-8880 [Office], 713/468-8883 [Fax]

# SWITCHING PROCESSOR THREADS DURING LONG LATENCIES

## Background

Processors, such as modern high-performance processors, are designed to execute a large number of instructions per clock cycle. Certain instructions produce a result only after a potentially large number of cycles. Such instructions may be known as "long latency" instructions, as a long time interval exists between the time an instruction is delivered and when it is executed. A long latency may occur, for example, when data required by an instruction needs to be loaded from a high level of memory. Such a load operation therefore may have a "load-use" penalty associated with it. That is, after a program issues such a load instruction, the data may not be available for multiple cycles, even if the data exists (i.e., "hits") in a cache memory associated with the processor.

Processors typically allow execution to continue while a long latency instruction is outstanding. Often, however, data is needed relatively soon (e.g., within several clock cycles) because insufficient work remains to be done by the processor without the requested data. Accordingly, a need exists to improve processor performance in such situations.

## Brief Description of the Drawings

FIG. 1 is a block diagram of a portion of a processor pipeline in accordance with one embodiment of the present invention.

FIG. 2 is a flow diagram of a method in accordance with one embodiment of the present invention.

FIG. 3 is a block diagram of a wireless device in accordance with one embodiment of the present invention.

## Detailed Description

Referring to FIG. 1, shown is a block diagram of a portion of a processor pipeline in accordance with one embodiment of the present invention. As shown in FIG. 1, pipeline 100 includes an instruction cache (I-cache) 110, an instruction fetch unit 120, an instruction decode unit 130, a register lookup 140, and reservation and execution unit 150, although the scope of the present invention is not limited in this regard.

While the type of processor which includes a pipeline in accordance with an embodiment of the present invention may vary, in one embodiment the processor may be a relatively simple in-order processor. In one embodiment, the processor may have a reduced instruction set computing (RISC) architecture, such as an architecture based on an Advanced RISC Machines (ARM) architecture. For example, in one embodiment a 32-bit version of an INTEL® XSCALE™ processor available from Intel Corporation, Santa Clara,

California may be used. However, in other embodiments the processor may be a different processor.

In one embodiment, I-cache 110 may be coupled to receive instructions from a bus interface unit of the processor. I-cache 110 may be used to store instructions, including instructions of multiple threads of a program. I-cache 110 may be coupled to provide instructions to instruction fetch unit 120. Alternately, instruction fetch unit 120 may receive instructions from a fill buffer (which may be within reservation and execution unit 150). Instruction fetch unit 120 may include, in certain embodiments, program counters for each thread to be executed on the processor, along with logic to sequence between the threads. In an embodiment in which out-of-order processing is implemented, instruction fetch unit 120 may include a branch target buffer that may be used to examine instructions fetched from I-cache 110 to determine whether any branching conditions exist.

As shown in FIG. 1, instruction decode unit 130 is coupled to receive fetched instructions from instruction fetch unit 120. Instruction decode unit 130 may be used to decode instructions by breaking more complex instructions into smaller instructions that may be processed faster. For example, in one embodiment instructions may be decoded into micro-operations (uops). However, in other embodiments other types of instructions may be decoded,

such as macro operations or another form of instruction. Additionally, it is to be understood that various instruction sets may be used, such as Reduced Instruction Set Computing (RISC) instructions or Complex Instruction Set Computing (CISC) instructions. Further, in one embodiment instruction decode unit 130 may decode CISC instructions to RISC instructions.

Still referring to FIG. 1, decoded instructions, including an identification of registers to be accessed, may be provided to register lookup 140. Register lookup 140 may be used to provide a physical register identification of a register in a register file unit. In such manner, registers may be assigned to each instruction. Also, register lookup 140 may, in certain embodiments, be used to stall the pipeline if register dependencies exist. Stalls are cycles in which the processor pipeline does not execute an instruction.

From register lookup 140, instructions may then proceed to reservation and execution unit 150 for scheduling execution of the instructions in the execution unit (or units) of the processor (e.g., an integer and/or floating point arithmetic logic unit (ALU)). In one embodiment, multiple execution units may be present. Such execution units may include a main execution pipeline, a memory pipeline and a multiply-accumulate (MAC) pipeline. In such an embodiment, the main execution pipeline may

perform arithmetic and logic operations, as required for data processing instructions and load/store index calculations, and may further determine conditional instruction execution. The memory pipeline may include a data cache unit to handle load and store instructions. The MAC pipeline may be used to perform multiply and multiply-accumulate instructions.

The above-described flow through pipeline 100 may describe normal operational flow. Such flow may occur when instructions do not require long latencies prior to execution, or in the absence of other conditions that may lead to processor stalls.

However in accordance with various embodiments of the present invention, when certain predetermined events or conditions occur, such as an instruction that may require a long latency prior to execution, a feedback loop 125 from instruction decode unit 130 to instruction fetch unit 120 may be activated to cause instruction fetch unit 120 to prepare to switch threads and accordingly fetch instructions for the new thread.

In one embodiment, identifications of predetermined conditions may be stored in instruction decode unit 130, although the scope of the present invention is not limited in this regard. For example, the identifications may be stored in a lookup table or other storage within instruction decode unit 130. In this manner, when an

instruction is received by instruction decode unit 130, it may be analyzed against entries in the lookup table to determine whether the instruction corresponds to one of the predetermined conditions. If so, a feedback signal on feedback loop 125 may be activated. Alternately, logic in instruction decode unit 130 may be used to detect for the presence or occurrence of a predetermined condition. In still other embodiments, microcode in instruction decode unit 130 may determine the presence or occurrence of a predetermined condition.

While the predetermined conditions may vary in different embodiments, an instruction that may require a long latency prior to execution may be considered to be a predetermined condition. As used herein, the term "long latency" means a time period between receipt of an instruction and execution of the instruction that causes the processor to suffer one or more stalls. Thus a long latency period may be several cycles or may be hundreds of cycles, depending on the ability of the processor to perform other instructions in the latency period. For example, a load instruction that requires the obtaining of information from system memory (e.g., as on a cache miss) may require hundreds of cycles, while a load instruction that obtains data from a cache (such as a level 1 (L1) cache) closely associated with the processor may require fewer than ten cycles. In certain embodiments, both load

instructions may be considered long latency instructions, as a processor may suffer stall cycles before data is ready for processing.

Thus a load instruction is an example instruction that may cause a long latency and thus may be considered a predetermined condition. While the latency caused by a load instruction may vary depending on the level of memory at which the data is obtained, the presence of a load instruction itself, regardless of actual latency, may be sufficient to cause a feedback signal in accordance with an embodiment of the present invention. In other words, a feedback signal in accordance with an embodiment of the present invention may be based on stochastic models. That is, the predetermined conditions may be selected based on a knowledge that the conditions may, but need not necessarily, cause a latency that leads to pipeline stalls.

Other examples of instructions that may be considered to be a predetermined condition may include store instructions and certain arithmetic instructions. For example, a floating point divide operation may be a condition that causes a feedback signal. In addition, other operations which require accessing a memory subsystem may be a condition causing a feedback signal to be initiated.

When a predetermined condition is detected, instructions of another thread may be fetched and executed

so that few or no stall cycles occur. Thus in certain embodiments, performance may be significantly increased in multi-thread contexts. At the same time, no performance difference occurs in a single thread context, because during single thread operation, instructions pass directly through pipeline 100, as discussed above.

Referring now to FIG. 2, shown is a flow diagram of a method in accordance with one embodiment of the present invention. While FIG. 2 relates specifically to a method for processing a long latency instruction, it is to be appreciated that the flow of FIG. 2 may be applicable to any predetermined condition. As shown in FIG. 2, method 200 may start (oval 205) and fetch an instruction of a current thread (block 210). Next, the instruction may be decoded (block 220). After decoding, it may be determined whether a predetermined condition has been met. For example, it may be determined whether the instruction is one of the predetermined conditions, for example, because it may cause a long latency (diamond 230). Such a determination may be made in instruction decode unit 130, in one embodiment.

If it is determined that the instruction may not have a long latency (i.e., a predetermined condition has not occurred), the instruction may be executed (block 240) and a next instruction of the current thread may be fetched (block 210).

If instead it is determined that a long latency may result (i.e., a predetermined condition has occurred), the pipeline may be prepared to switch threads (block 250). In one embodiment, preparation to switch threads may include executing a remaining one or several instructions of the first thread prior to the thread switch. In such manner, additional instructions may be performed without a processor stall. More so, because such instructions are already present in the processor pipeline, they need not be flushed. Thus instructions of a first thread may continue to be processed while the second thread is prepared for execution. For example, an embodiment executed in the processor pipeline 100 of FIG. 1 may have a two pipeline-stage delay caused by the feedback signal on feedback loop 125. This delay may thus allow two additional instructions of the first thread to be performed before the second thread begins execution.

Preparing to switch threads may further include setting instruction fetch unit 120 to fetch an instruction of the new thread. For example, a program counter for the second thread within instruction fetch unit 120 may be selected and used to fetch the next instruction of the thread.

Next, the threads may be switched (block 260). In the embodiment of FIG. 2, such thread switching may cause instruction fetch unit 120 to obtain an instruction of the

new thread from I-cache 110 (block 210). Additionally, various control registers and other information corresponding to the new thread may be loaded into different registers and portions of the processor pipeline to allow execution of instructions of the new thread.

Referring now to Table 1, shown is an example of execution of a code portion in accordance with an embodiment of the present invention:

TABLE 1

```
Thread0:
     ADD R2
     LDR to R0
     SUB R3
     XOR R4

Thread1:
     MUL  R3
     LDR to R7
     ADD  R1
     AND  R4

Thread0:
     ADD R0
     …
```

As shown in Table 1, a first thread (Thread 0) may be executing in a processor pipeline. When a long latency instruction is detected by instruction decode unit 130 (i.e., the Load to register 0 (LDR to R0) instruction), a signal may be sent on feedback line 125 to instruction fetch unit 120 to prepare for a new thread. At the same time, one or more additional instructions of the first

thread may be executed in the processor pipeline (e.g., SUB R3 and XOR R4).

Then as shown in Table 1, a second thread (i.e., Thread 1) may be switched to and begun activation. As with the first thread, the second thread may begin executing instructions and continue until a long latency instruction is encountered (i.e., the Load to register 7 (LDR to R7) instruction). As above, at the detection of a long latency instruction, preparation may be made to switch threads again, while at the same time processing one or more additional instructions of the current thread.

Finally as shown in Table 1, the original thread (i.e., Thread 0) may again be switched to and begun execution. While shown and discussed in Table 1 as having two threads of operation, it is to be understood that in other embodiments multithread operations may encompass more than two threads.

Embodiments of the present invention may be implemented in code and may be stored on a storage medium having stored thereon instructions which can be used to program a system, such as a wireless device to perform the instructions. The storage medium may include, but is not limited to, any type of disk including floppy disks, optical disks, compact disk read-only memories (CD-ROMs), compact disk rewritables (CD-RWs), and magneto-optical disks, semiconductor devices such as read-only memories

(ROMs), random access memories (RAMs), erasable programmable read-only memories (EPROMs), flash memories, electrically erasable programmable read-only memories (EEPROMs), magnetic or optical cards, or any type of media suitable for storing electronic instructions, including programmable storage devices.

FIG. 3 is a block diagram of a wireless device with which embodiments of the invention may be used. As shown in FIG. 3, in one embodiment wireless device 500 includes a processor 510, which may include a general-purpose or special-purpose processor such as a microprocessor, microcontroller, application specific integrated circuit (ASIC), a programmable gate array (PGA), and the like. Processor 510 may include a feedback loop in accordance with one embodiment of the present invention and may be programmed to switch threads when a predetermined condition occurs. Processor 510 may be coupled to a digital signal processor (DSP) 530 via an internal bus 520. In turn, DSP 530 may be coupled to a flash memory 540. As further shown in FIG. 3, flash memory 540 may also be coupled to microprocessor 510, internal bus 520, and peripheral bus 560.

As shown in FIG. 3, microprocessor 510 may also be coupled to a peripheral bus interface 550 and a peripheral bus 560. While many devices may be coupled to peripheral bus 560, shown in FIG. 3 is a wireless interface 570 which

is in turn coupled to an antenna 580. In various embodiments antenna 580 may be a dipole antenna, helical antenna, global system for mobile communication (GSM) or another such antenna.

Although the description makes reference to specific components of device 500, it is contemplated that numerous modifications and variations of the described and illustrated embodiments may be possible.

While the present invention has been described with respect to a limited number of embodiments, those skilled in the art will appreciate numerous modifications and variations therefrom. It is intended that the appended claims cover all such modifications and variations as fall within the true spirit and scope of this present invention.